PARALELIZACIÓN DEL PROGRAMA DE MONTE CARLO KENO-Va

J. RAMÓN - J. PEÑA

Javier RAMÓN CAMARMA es licenciado en Ciencias Físicas, especialidad de Física Fundamental, por la Universidad Complutense de Madrid (1985). En 1990 ingresó en el Consejo de Seguridad Nuclear en la Unidad de Análisis del Núcleo. En 1993 pasó al Departamento de Informática como Jefe de Servicio de Cálculo y Sistemas, dedicándose fundamentalmente a la adaptación y optimización de códigos de cálculo y al desarrollo de interfaces de usuario.

Jorge PEÑA GUTIÉRREZ es Ingeniero Industrial por la Escuela Técnica Superior de Ingenieros Industriales de Madrid (1973). Desarrolló sus actividades en el campo de la Tecnología Nuclear en la Junta de Energía Nuclear. En la actualidad trabaja en el Consejo de Seguridad Nuclear como Jefe del Área de Sistemas Informáticos y es responsable de las actividades de aplicaciones técnicas y cálculo avanzado en el Departamento de Informática.

INTRODUCCIÓN

El Consejo de Seguridad Nuclear ha iniciado una actividad cuyo objetivo es la paralelización de los programas de cálculo más utilizados en el organismo y que requieren un tiempo de ejecución grande. Con objeto de adquirir los conocimientos prácticos en los modelos de programación que han de utilizarse en esta actividad, se ha seleccionado el programa de Monte Carlo KENO-Va [1] debido al paralelismo inherente de este tipo de programas.

Se han desarrollado dos versiones del programa KENO: una para computadores de memoria compartida y otra para sistemas de memoria distribuida. La versión de memoria compartida utiliza directivas del compilador FORTRAN para declarar aquellas partes del programa que pueden ejecutarse en paralelo. La versión de memoria distribuida ha requerido la generación de diversos procesos concurrentes, que se comunican mediante mensajes, y que trabajan bajo un esquema denominado maestro-esclavo.

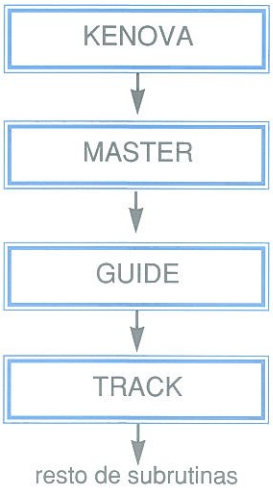
Este artículo pretende describir los cambios fundamentales que han tenido que realizarse y comentar los aspectos más relevantes relacionados con la aplicación de estas técnicas de programación. En la referencia 2 se describe exhaustivamente el trabajo realizado en el desarrollo de la versión para memoria compartida.

El artículo se organiza de la forma siguiente: primeramente se describen los cambios de carácter general que se han realizado a la estructura del programa KENO; seguidamente se describen las modificaciones concretas efectuadas en las versiones de memoria compartida y distribuida. A continuación se dedica un apartado concreto a la generación de números aleatorios. Por último se presentan los resultados obtenidos de la ejecución de diversos casos y las conclusiones.

CAMBIOS A LA ESTRUCTURA DEL PROGRAMA KENO

La descripción detallada de la lógica del programa está contenida en la referencia 1, en este documento se van a comentar aquellas partes más importantes del programa.

La secuencia de ejecución es la siguiente:



KENOVA: es el programa principal
MASTER: es la subrutina que controla la ejecución global del programa
GUIDE: es la subrutina que controla las distintas generaciones de neutrones
TRACK: es la subrutina que simula la trayectoria de los neutrones

Las historias de los neutrones pertenecientes a una misma generación son independientes, por lo tanto pueden seguirse independientemente. El bucle que gobierna la simulación de dichos neutrones se realiza en la subrutina TRACK, y puede ejecutarse en paralelo. El seguimiento de los neutrones de una generación permiten calcular la fuente para la generación siguiente, por consiguiente, los cálculos generacionales han de hacerse secuencialmente.

Los cambios que hay que realizar para poder ejecutar en paralelo el programa dependen del modelo de programación.

En el modelo de **memoria compartida** se utilizan directivas del compilador FORTRAN para declarar aquellas partes del programa que pueden ejecutarse en paralelo. Es necesario tener presente los eventos que habitualmente inhiben o dificultan la paralelización en sistemas de memoria compartida:

- Múltiples entradas y salidas en bucles
- Llamadas a funciones y subrutinas
- Sentencias de entrada/salida

• Dependencias de datos entre ciclos de un bucle
• Variables compartiendo las mismas direcciones vía *equivalence*

Concretamente en el caso del KENO, se ha utilizado una directiva para que actúe sobre el bucle que efectúa el seguimiento de los diferentes neutrones de una generación. Además, se necesitaron otros cambios para modificar el ámbito de ciertas variables (globales o locales) y algunas estructuras de datos.

En el modelo de **memoria distribuida** la situación es muy diferente. Se adopta un modelo de programación basado en la ejecución de varios procesos simultáneos en diferentes procesadores, comunicándose mediante el envío de mensajes entre ellos. De entre las formas posibles de establecer un modelo de este tipo, el más idóneo es el conocido por la denominación maestro-esclavo. Especial atención hay que dedicar en este modelo a la sincronización entre procesos.

En modelos maestro-esclavo un proceso principal, el maestro, crea tantos procesos secundarios idénticos, los esclavos, como sean necesarios. El proceso maestro prepara los datos para los procesos esclavos, se los envía y se mantiene a la espera de que dichos procesos concluyan su trabajo para recibir los resultados y procesarlos. La comunicación se mantiene solo entre el proceso maestro y los procesos esclavos; éstos no se comunican entre sí. La cantidad de trabajo que realiza cada proceso esclavo puede ser diferente, de forma que, realizando una distribución adecuada del trabajo, el balance de carga en todo el sistema puede estar equilibrado. En la figura I se presenta una visión esquemática de este modelo.

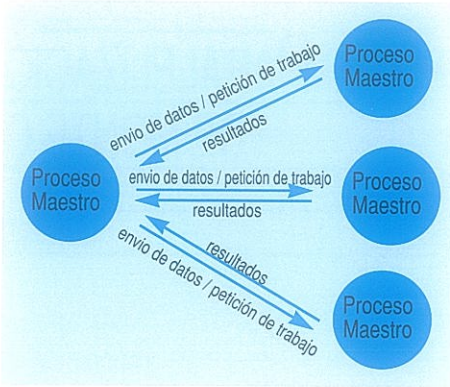
Para los trabajos realizados en el CSN con KENO se eligió la librería PVM [3] para realizar la comunicación entre procesos mediante el paso de mensajes.

VERSION PARA MEMORIA COMPARTIDA

Como ya se ha indicado anteriormente, GUIDE es la subrutina que controla las historias de los neutrones. Contiene el bucle sobre generaciones y llama, dentro de dicho bucle, a TRACK que es la subrutina que realiza el seguimiento de las trayectorias de los neutrones.

Originalmente la subrutina GUIDE está estructurada de esta forma:

```
SUBROUTINE GUIDE
Bucle sobre generaciones
  Preparación del banco de neutrones
  CALL TRACK ──► SUBROUTINE TRACK
                   Bucle sobre historias
                     Realización del seguimiento
                   Fin del bucle sobre historias
                 ◄── RETURN
  Procesamiento del banco de fisiones
Fin del bucle sobre generaciones
```



F I *Representación esquemática del paradigma maestro-esclavo*

Los principales cambios introducidos son los siguientes:

• Se establece dentro del programa el tamaño de los lotes de neutrones. Dado que el número de historias de neutrones por generación es un dato de entrada, éstas pueden distribuirse en lotes previamente a comenzar la simulación. Cada lote se ejecuta en un procesador. El número de lotes y el correspondiente tamaño de lote (granularidad) han de elegirse de forma que se logre la mayor velocidad de ejecución (tiempo real). En el caso del desarrollo efectuado en el Convex C3440 el número óptimo de lotes se ha establecido en 23.

• Se crea una nueva subrutina, TRACK_SLAVE, que procesa los neutrones incluidos en un lote. En esta versión TRACK prepara los índices de los lotes, llama a TRACK_SLAVE en paralelo y procesa los resultados obtenidos de las diversas ejecuciones.

• Se modifican varias estructuras de datos para adecuarse a la nueva organización del programa:

- variables globales que pasan a ser locales,
- vectores que tienen que modificarse para contener datos intermedios que tienen que se procesados por la nueva subrutina TRACK,
- se crea un nuevo *common*, otros se modifican,
- se cambia la gestión de números aleatorios

Como consecuencia de la nueva estructuración la subrutina GUIDE queda como sigue:

```
SUBROUTINE GUIDE
Bucle sobre generaciones
  Preparación del banco de neutrones
  CALL TRACK ──► SUBROUTINE TRACK
                   Preparación de los índices de los lotes
                   Bucle sobre lotes
                     CALL TRACK - SLAVE ──► SUBROUTINE TRACK - SLAVE
                                              Bucle sobre historias
                                                Seguimiento
                                              Fin del bucle historias
                                            ◄── RETURN
                   .....
                   Fin del bucle sobre lotes
                 ◄── RETURN
  Procesamiento del banco de fisiones
Fin del bucle sobre generaciones
.....
```

En un sistema multiprocesador de memoria compartida las *threads* asociadas al programa se generan dinámicamente de acuerdo con la carga instantánea del sistema. Si no se controla la generación de números aleatorios puede ocurrir que dos simulaciones consecutivas den resultados diferentes. Más adelante se comentará cómo se ha tenido que efectuar la gestión de los números aleatorios.

VERSION PARA MEMORIA DISTRIBUIDA

Como ya se ha indicado anteriormente, la versión para sistemas de memoria distribuida se ha realizado utilizando el modelo maestro-esclavo, de forma que los procesos maestro y esclavos pudiesen ejecutarse concurrentemente y de forma asíncrona. En el desarrollo se utilizó la librería PVM [3] de comunicaciones.

Las subrutinas más relevantes que forman los procesos maestro y esclavo son las siguientes:

Proceso maestro

KENOVA : Programa principal

↓

MASTER : Crea los procesos esclavos y les envía datos genéricos

↓

GUIDE : Envía datos relacionados con la generación y el correspondiente banco de neutrones acada proceso esclavo.
Procesa datos de fisión de cada proceso esclavo y crea un banco de fisión maestro

↓

resto subrutinas

Proceso esclavo

KENOVA-SLAVE : Recibe parte de los datos genéricos

↓

POINTERS : Recibe el resto de los datos genéricos

↓

GUIDE : Recibe datos asociados a la generación

↕

TRACK : Realiza el seguimiento de las historias

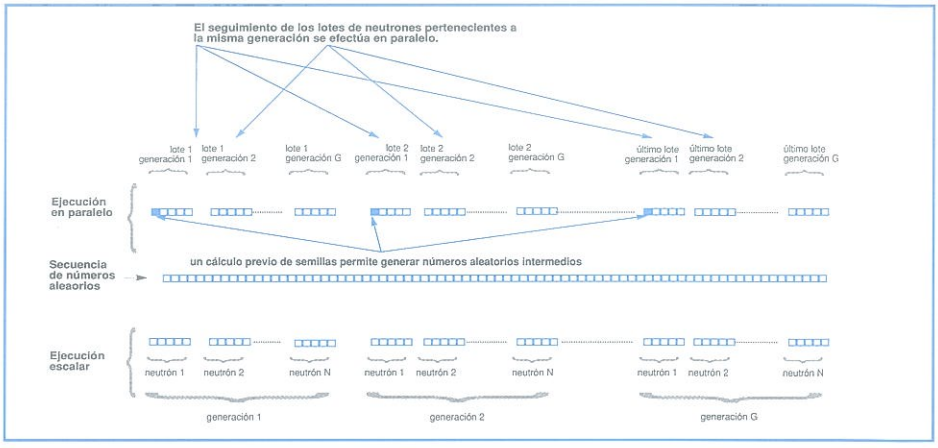
El proceso maestro envía a los procesos esclavos colaboradores una señal para notificarles el final de la ejecución.
Al igual que en el caso de memoria compartida, la gestión de números aleatorios se efectuó de una forma específica.

GENERACION DE NUMEROS ALEATORIOS

Para conseguir que los resultados de las ejecuciones con KENO puedan ser reproducidos es necesario usar técnicas específicas en la selección (muestreo) de neutrones de fuente y el seguimiento de las trayectorias de cada uno de ellos. Si no se generan adecuadamente la serie de números aleatorios puede suceder que dos ejecuciones consecutivas conduzcan a valores diferentes.
El método consiste en el cálculo previo de semillas a partir de las cuales se puedan generar, independientemente, grupos de números aleatorios todos ellos pertenecientes a una misma secuencia. El tamaño de cada grupo de números aleatorios (distancia entre semillas) se ha establecido en 1280 para los cálculos que se han realizado. El método de obtención de las semillas se explica en la referencia [4].
En la figura II se representa esquemáticamente la generación y las asignación de los números aleatorios a las diferentes formas de efectuar el seguimiento de las trayectorias, tanto secuencial (uno a uno) como en paralelo (por lotes).

RESULTADOS OBTENIDOS

Con objeto de verificar las dos versiones desarrolladas del programa KENO, se han ejecutado diversos casos ejemplo y reales. Los resultados se presentan en las tablas I y II.
En la realización de la versión en memoria compartida se ha utilizado el Convex C3440, un tetra-procesador con 1 Gbyte de memoria compartida. Los factores de aceleración obtenidos en la ejecución de los casos ejemplo fue de 3,5. En un caso real de un contenedor de combustible irradiado se obtuvo una aceleración de 3,6.
La versión de KENO en un sistema distribuido se desarrolló en un *cluster* de 6 HP9000/735 con comunicaciones FDDI ubicado en el CEDEX, utilizando la versión propietaria de Convex de la librería de paso de mensajes PVM. La versión se preparó de tal forma que el programa maestro se ejecutaba en un único nodo, y 5 procesos esclavos se instalaron en otros tantos procesadores. El mismo caso real de contenedor mencionado anteriormente se ejecutó con un factor de aceleración de 3,6.
Es importante mencionar que el tiempo de ejecución (en modo escalar) en el *cluster* fue de unos 12 minutos; aproximadamente la décima parte del tiempo de ejecución invertido en el Convex. En otros casos de mayor consumo de tiempo de ejecución, los rendimientos que se obtendrían en el *cluster* serían mayores.



F II - ***Visión esquemática de la gestión de números aleatorios.*** *En ejecuciones en modo escalar, los números aleatorios se generan de forma secuencial. En ejecuciones en modo paralelo, la secuencia de números aleatorios se divide en tantas partes como procesos simultáneos o"threads" existan. Un cálculo previo de semillas permite obtener secuencialmente los números aleatorios para todas las generaciones que estudia cada proceso.*

Los factores de aceleración obtenidos en el Convex son muy próximos a los teóricos. No así en el sistema distribuido, donde no se ha invertido apenas esfuerzo en optimizar la configuración del sistema de cálculo para obtener un rendimiento mayor.

CONCLUSIONES

Se ha elegido el programa de Monte Carlo KENO para adquirir el conocimiento de las técnicas de paralelismo mediante el empleo de modelos de programación específicos para sistemas de sistemas de memoria compartida y distribuida. La elección de un programa de Monte Carlo se ha debido, fundamentalmente, a su inherente paralelismo.

La versión para memoria compartida utiliza directivas del compilador FORTRAN de Convex. La versión de memoria distribuida se ha desarrollado utilizando la representación maestro-esclavo y un sistema de comunicación basado en el paso de mensajes.

T I
Resultados obtenidos en un Convex C3440 para algunos casos ejemplo incluidos en el paquete de distribución del programa de la NEA

Casos Ejemplo						
CASO	tiempo CPU (min: seg)	tiempo sistema (min: seg)	tiempo real (min: seg)	uso CPU 4cpu=100%	factor de concurrencia	Acelaración
#1	0:29.42	0:02.67	0:14.02	57.2 %	2.64	**2.29**
#2	0:34.41	0:02.59	0:15.86	58.3 %	2.59	**2.33**
#3	4:26.01	0:05.37	1:25.58	79.3 %	3.34	**3.17**
#4	4:19.14	0:05.32	1:25.69	77.2 %	3.32	**3.09**
#5	0:50.44	0:03.72	0:20.54	65.9 %	2.95	**2.64**
#6	0:16.92	0:02.41	0:09.83	49.2 %	2.23	**1,97**
#7	0:29.41	0:02.53	0:13.54	59.0 %	2.62	**2.36**
#8	0:22.70	0:02.44	0:11.37	55.3 %	2.50	**2.21**
#9	0:28.51	0:02.05	0:11.25	67.9 %	3.05	**2.72**
#10	0:29.68	0:02.75	0:14.25	56.9 %	2.54	**2.28**
#11	0:15.23	0:01.75	0:07.65	55.5 %	2.54	**2.22**
#12	1:08.95	0:03.92	0:28.04	65.0 %	2.87	**2.60**
#13	0:00.67	0:01.30	0:02.01	24.5 %	1.00	**0.98**
#14	0:25.05	0:02.46	0:11.96	57.5 %	2.58	**2.30**
#15	10:26.31	0:08.99	3:09.44	83.8 %	3.50	**3.35**
#16	1:15.06	0:04.04	0:34.17	72.5 %	3.18	**2.90**
#17	2:02.56	0:03.22	0:43.46	72.4 %	3.07	**2.90**
#18	4:20.75	0:05.26	1:25.00	78.2 %	3.31	**3.13**
#19	1:15.09	0:03.96	0:29.02	68.1 %	3.00	**2.72**
#20	1:10.07	0:03.62	0:27.87	66.1 %	2.89	**2.64**
#21	6:00.38	0:05.90	1:46.46	86.0 %	3.61	**3.44**
#22	0:58.20	0:02.75	0:21.96	69.4 %	3.03	**2.78**
#23	0:41.75	0:02.61	0:17.15	64.7 %	2.85	**2.70**
#24	0:42.55	0:02.71	0:17.54	64.5 %	2.87	**2.58**
#25	0:43.08	0:02.61	0:17.83	64.1 %	2.84	**2.56**

Pasa a la página 31.